All Revisions Search

Update: Replaced latest flag with all_revisions, which does the opposite. When all_revisions=true, we return all revisions. When all_revisions=false (the default) we only return the latest.

Goal

The CMR should support returning of all revisions of a concept, not just the most recent. This allows users to write clients to manage and compare these different revisions.

Traceability

CMR-1805 - JIRA project doesn't exist or you don't have permission to view it.

CMR-1747 - JIRA project doesn't exist or you don't have permission to view it.

CMR-1416 - JIRA project doesn't exist or you don't have permission to view it.

CMR-1418 - JIRA project doesn't exist or you don't have permission to view it.

CMR-1776 - JIRA project doesn't exist or you don't have permission to view it.

CMR-1777 - JIRA project doesn't exist or you don't have permission to view it.

CMR-1781 - JIRA project doesn't exist or you don't have permission to view it.

Design

After some initial whiteboard design, we rejected approaches involving doing everything in SQL or doing ACL filtering after pulling everything back. The approach we are going to prototype involves adding a new index to Elasticsearch that will hold all revisions of the collections.

Currently collections are stored in a single index as shown in Table 1 for providers PROV1 and PROV2. Only the most recent revision of each collection is stored according to Elasticsearch's `_version` policy. Under the new approach, we will retain the current index structure for serving normal requests. For requests for older revisions we will use a single index like the one shown in Table 2.

Search will occur as usual, but a new parameter, `all_revisions`, will be used to determine which index to search. When `all_revisions=false` (the default) then search will occur as normal. When `all_revisions=true`, search will use the All Revisions index and return all revisions.Which formats we support in the response is an open question - we need to talk to MMT team about this. (JASON: We should do that very soon. Today or tomorrow. Schedule a meeting.)

Existing Collection Index
id:C100000-PROV1 _version:5
id:C100000-PROV2 _version:2
id:C100001-PROV1 _version:1
id:C100001-PROV2 _version:1
id:C100002-PROV1 _version:3

Table 1 - Collection index

All Revisions index
id:C100000-PROV1,3 _version:3
id:C100000-PROV1,4 _version:4
id:C100000-PROV1,5 _version:5
id:C100001-PROV1,1 _version:1
id:C100002-PROV1,1 _version:1
id:C100000-PROV2,1 _version:1
id:C100000-PROV2,2 _version:2
id:C100001-PROV2,3 _version:1

Table 2 - All Revisions index for all collection revisions

For the All Revisions index, we will store every revision of each collection (approximately 500K revisions in production now). In order to do this without violating Elasticsearch's `_version` policy we will need to use a different `id` field than the current indices. This field will be generated by combining `concept-id` with `revision-id`. For example `concept-id + "," + revision-id`.

Populating the new index will happen in two phases

1. A bootstrap phase to load the concept revisions from metadata db. This should be implemented as a job that can be re-run periodically to maker sure the index is up-to-date. (JASON: We will need to start indexing into this index and then run bootstrap. Otherwise bootstrap will miss somethings just being added.)
2. Ongoing indexing via a new channel on our index queue as shown in Figure 1.

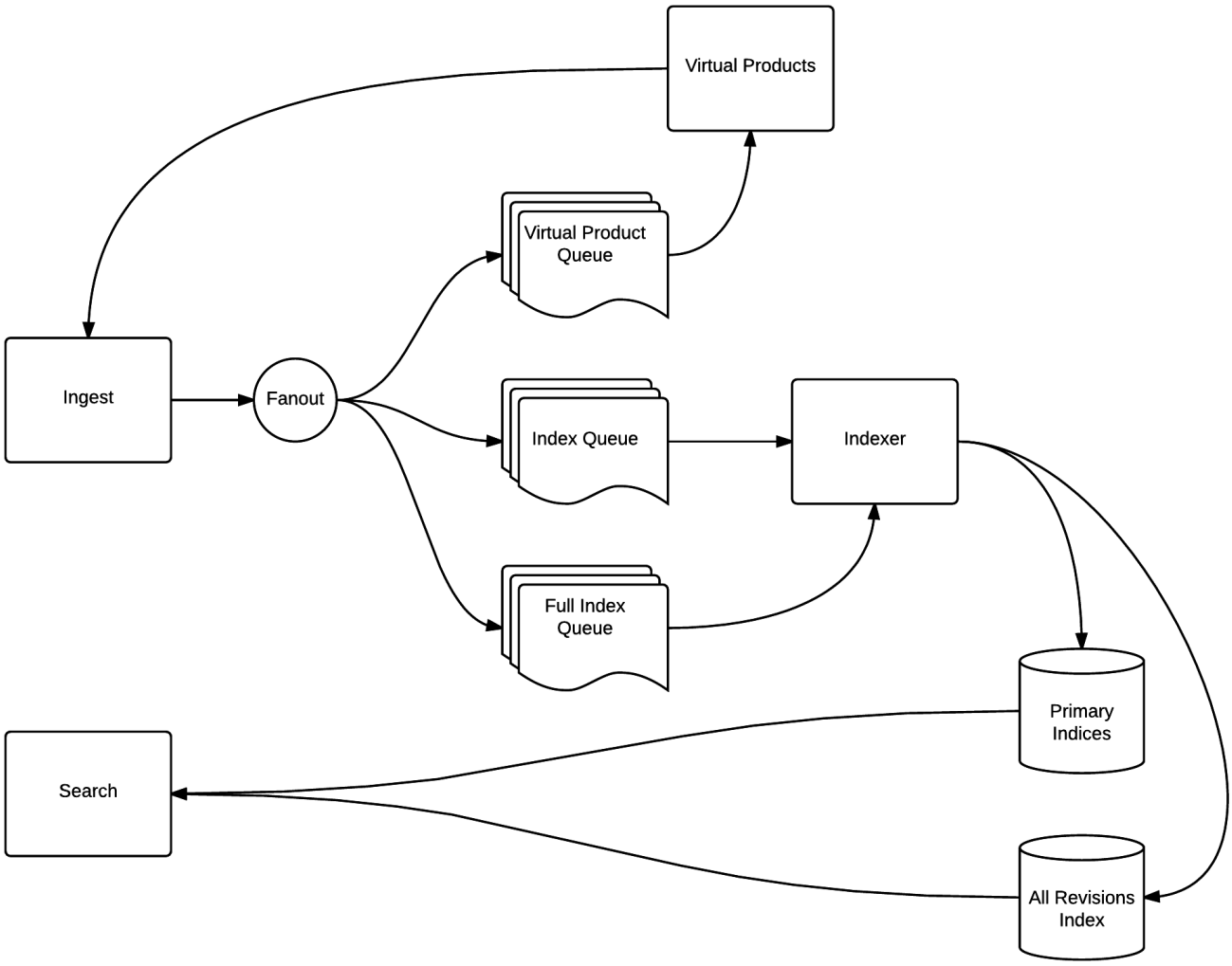

Figure 1 - Concept revision search with the new index

This buys us several thing basically for free:

1. ACLs
2. Paging support
3. Response format support
4. Validations

JASON: Add information here about sorting results. We need to change the default sorting. Whenever you select to sort by anything in the search app like provider id we append another sort by concept id. This guarantees that two searches with a sort by provider id would return items in the same order since provider id isn't enough by itself to fully dictate the order of all items. (See cmr.search.data.query-to-elastic/query->sort-params) Given that we're adding a new index which has duplicate items with the same concept id we need to sort by concept-id and revision id. I think we should add revision id as an indexed field in both the existing collection index and the new all revision index. Then we should always sort by revision id in addition to concept id for collection queries. I don't think we should include granules with this change because it would have performance implications. The query->sort-params will need to change slightly then based on the concept type.

An additional exchange/queue will be added for concept revision cleanup. This will be utilized by the metadata-db concept-revision-cleanup job to request deletions from the All Revisions index as shown in Figure 2.

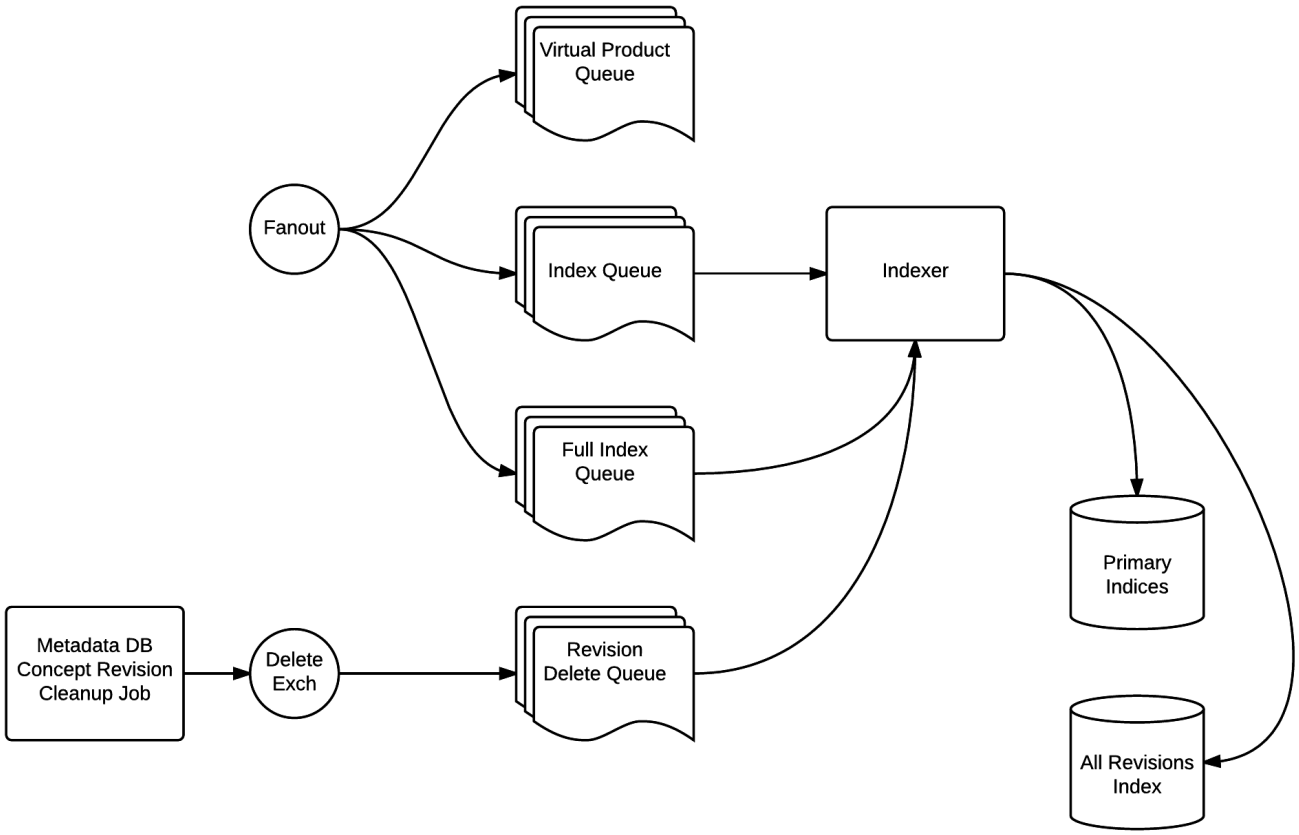

Figure 2 - Concept revision cleanup

Code Changes

- Initial prototype
    - Indexing
        - Add new index to index set.
        - Modify indexer code to generate combination concept-id/revision-id document ids and insert into new index.
        - Add code to handle tombstones.
    - Searching
        - Remove existing /concept-revisions endpoint and consolidate with primary find concepts route. (Not sure about this one)
        - Add new "latest" parameter and code changes to use it to trigger concept revision search.
    - Fix anything that breaks
- Full implementation
    - Bootstrapping to index existing concepts
    - Old revision cleanup
        - Use additional queue for issuing deletes via metadata-db old-revision-cleanup job

Tests

- Existing tests should be good enough for prototype
    - Modify test utils to hit consolidated endpoint
    - Modify tests for multiple revisions to pass `latest=false` in params
    - Modify expected "find latest" logic in existing tests that explicitly test it (these are commented)
- Add tests passing explicit `latest=true`
- Unit test for id generation

Questions / Assumptions

1. Should we use a separate queue for All Revisions index requests? (assume yes - might have benefits for bootstrapping) - Another point - two queues allows separate retry logic - if one index operation fails it will get retried, not both.
2. Should we use a separate listener or modify the existing listener to handle both types of requests? (assume no)
3. Index everything in the same way that we do now (same fields, same code, different index and id).
4. Tombstones will be converted to elastic documents and indexed. Use empty fields for everything except the fields in metadata-db. (JASON: There are some acl implications here since some acls grant access based on data in the metadata that a tombstone won't have. I think we'll be ok though since people searching this way will have been granted access to all data within a provider or by the id.)
5. Index will be created like all the rest by adding it to the index set.
6. Should we use the queue for the initial prototype? (assume yes - might actually be more complicated to make a direct call)
7. Is it OK for a user to be able to retrieve old revisions subject to current ACLs?
8. UMM-JSON is the response format (application/umm+json)
9. Use all_revisions flag instead of latest.

Error rendering macro 'pageapproval' : null